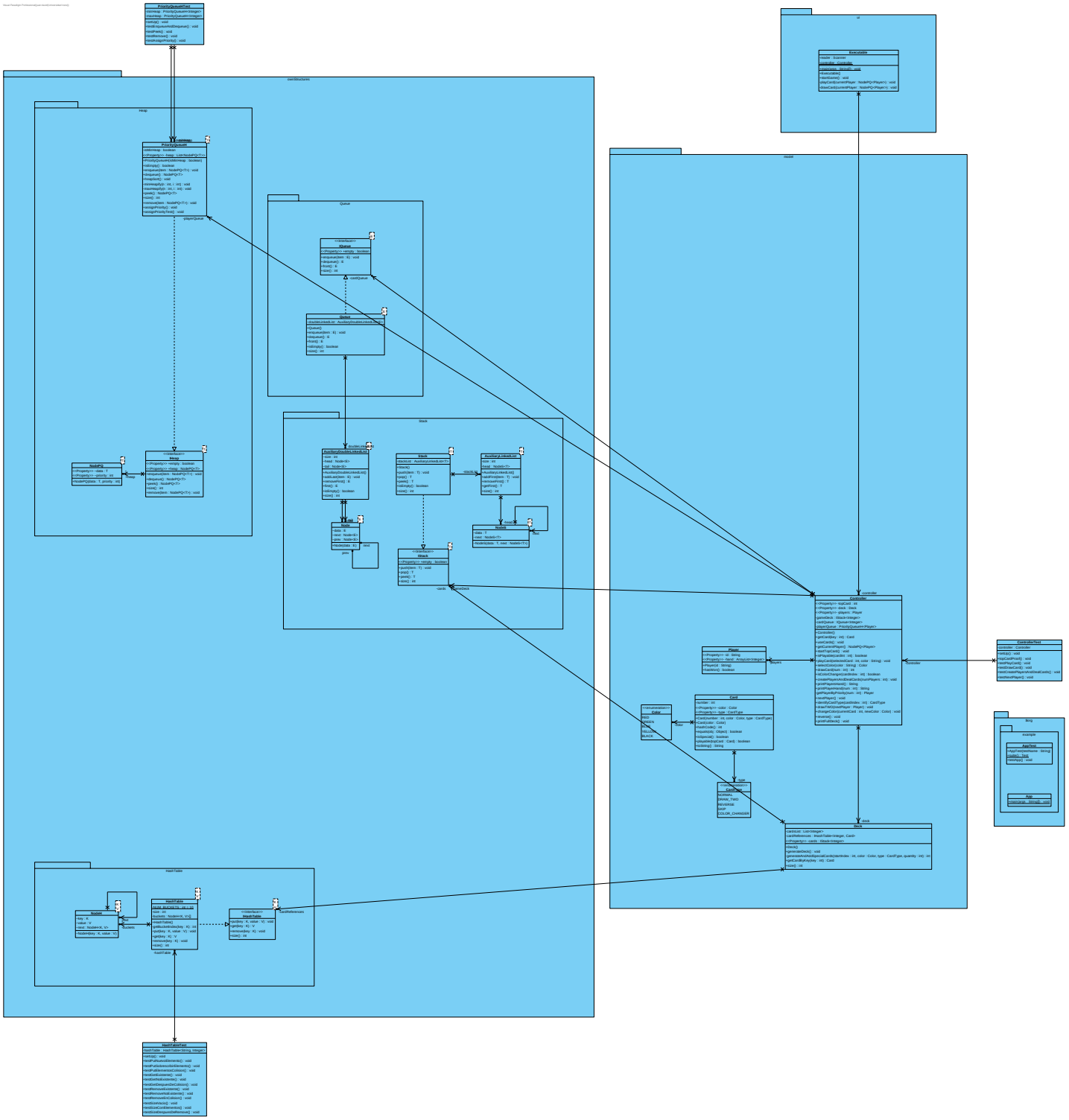
Visual Paradigm Professional(uat-uat4)(nonedate:max)
PriorityQueueInt
+minHeap : PriorityQueue<Integer>
+maxHeap : PriorityQueue<Integer>
+main(arg) : void
+testEnQueueAndDequeue() : void
+testPeek() : void
+testRemove() : void
+testAssignPriority() : void
dsaAlgo
dsaAlgo::dsaAlgo::dataStructures
Heap
PriorityQueue
+isMinHeap : boolean
#<Property>> - heap : List<NodePQ<T>>
+PriorityQueue(isMinHeap : boolean)
+isEmpty() : boolean
+enqueue(item : NodePQ<T>) : void
+dequeue() : NodePQ<T>
+heapify() : void
#swapMap(p1i : int, i : int) : void
#swapHeap(p1i : int, i : int) : void
+peek() : NodePQ<T>
+size() : int
+remove(item : NodePQ<T>) : void
+assignPriority() : void
+assignPriority(key) : void
playerQueue
<<interface>>
Heap
+<Property>> +empty() : boolean
+<Property>> +heap : NodePQ<S>
+enqueue(item : NodePQ<T>) : void
+dequeue() : NodePQ<T>
+peek() : NodePQ<T>
+size() : int
+remove(item : NodePQ<T>) : void
NodePQ
+<Property>> +data : T
+<Property>> +priority : int
+NodePQ(data : T, priority : int)
Map
Queue
<<interface>>
Queue
+<Property>> +empty() : boolean
+enqueue(item : E) : void
+dequeue() : E
+front() : E
+size() : int
cardQueue
Deque
+doubleLinkedList : AuxiliaryDoubleLinkedList<E>
+Deque()
+enqueue(item : E) : void
+dequeue() : E
+front() : E
+isEmpty() : boolean
+size() : int
Stack
doubleLinkedList
AuxiliaryDoubleLinkedList
-size : int
-head : Node<E>
-tail : Node<E>
+AuxiliaryDoubleLinkedList()
+addLast(item : E) : void
+removeFirst() : E
+first() : E
+isEmpty() : boolean
+size() : int
Node
-data : E
-next : Node<E>
-prev : Node<E>
+Node(data : E)
next
prev
Stack
-stackList : AuxiliaryLinkedList<T>
+Stack()
+push(item : T) : void
+pop() : T
+peek() : T
+isEmpty() : boolean
+size() : int
stackList
AuxiliaryLinkedList
-size : int
-head : Node<T>
+AuxiliaryLinkedList()
+addFirst(item : T) : void
+removeFirst() : T
+peekFirst() : T
+size() : int
head
Node
-data : T
-next : Node<T>
+Node(data : T, next : Node<T>)
next
<<interface>>
IStack
+<Property>> +empty() : boolean
+push(item : T) : void
+pop() : T
+peek() : T
+size() : int
cards
cardStack
HashTable
Node
-key : K
-value : V
-next : Node<K,V>
+Node(key : K, value : V)
next
buckets
HashTable
-MIN_BUCKETS : int = 16
-size : int
-buckets : Node<K,V>[]
+HashTable()
-getBucketIndex(key : K) : int
+put(key : K, value : V) : void
+get(key : K) : V
+remove(key : K) : void
+size() : int
-hashTable
<<interface>>
IHashTable
+put(key : K, value : V) : void
+get(key : K) : V
+remove(key : K) : void
+size() : int
hashTableRefactored
HashTableTest
hashTable : HashTable<String, Integer>
+setup() : void
+testPutNewElements() : void
+testPutExistingElements() : void
+testPutElementsInCollision() : void
+testGetExistingKey() : void
+testGetNotExistingKey() : void
+testGetSequentialCollision() : void
+testRemoveExistingKey() : void
+testRemoveNotExistingKey() : void
+testRemoveInCollision() : void
+testSizeVariety() : void
+testSizeConsiderDeleting() : void
+testSizeSequentialReference() : void
id
Executable
-model : IModel
-controller : IController
+execute(arg0 : String[]) : void
+IExecutable()
+startGame() : void
+playCardCurrentPlayer : NodePQ<IPlayer> : void
+drawCardCurrentPlayer : NodePQ<IPlayer> : void
model
controller
<<Property>> -topCard : int
<<Property>> -deck : Deck
<<Property>> -players : Player
-gameDeck : Stack<Integer>
-cardQueue : IQueue<Integer>
-playerQueue : PriorityQueueInt<IPlayer>
+Controller()
+getCardObj(v : int) : Card
+getCards() : void
+getPlayers(arg0 : NodePQ<IPlayer>)
+startTurn(arg0 : int) : void
+isPlayable(cardId : int) : boolean
+playCardsInHandCard : int, color : String) : void
+selectColor(color : String) : Color
+drawCard(num : int) : int
+isColorChanger(cardIndex : int) : boolean
+createPlayersAndDealCardsToPlayers : int) : void
+printPlayersHand() : String
+getPlayerPosition(p : int) : String
+getPlayerByPriority(num : int) : Player
+endPlayer() : void
+identifyCardType(cardIndex : int) : CardType
+drawTWOOnNextPlayer : IPlayer) : void
+changeColorLuminCard : int, newColor : Color) : void
+reverse() : void
+printPlayers() : void
Player
<<Property>> -id : String
<<Property>> -hand : Array<int>[]
+IPlayer(id : String)
+hasWon() : boolean
players
controller
ControllerTest
controller : Controller
+setUp() : void
+testCardPlayed() : void
+testPlayCard() : void
+testDrawCard() : void
+testCreatePlayersAndDealCards() : void
+testNextPlayer() : void
Card
-number : int
<<Property>> -color : Color
<<Property>> -type : CardType
+Card(number : int, color : Color, type : CardType)
+Card(color : Color)
+hashCode() : int
+equals(obj : Object) : boolean
+isSpecial() : boolean
+isMatchBetweenCard : Card) : boolean
+toString() : String
<<enumeration>>
Color
RED
GREEN
BLUE
YELLOW
BLACK
Color
type
<<enumeration>>
CardType
NUMBER
DRAW_TWO
REVERSE
SKIP
COLOR_CHANGER
deck
Deck
-cardsList : List<Integer>
-cardReferences : HashTable<Integer, Card>
<<Property>> -cards : Stack<Integer>
+Deck()
+generateDeck() : void
+generateAndDealSpecialCards(cardIndex : int, color : Color, type : CardType, quantity : int) : void
+getCardByKeyKey(key : int) : Card
+size() : int
log
example
AppTest
+AppTest(testName : String)
+suite() : Test
+testApp() : void
App
+main(args : String[]) : void